UNLIMITED

(2)

RSRE
MEMORANDUM No. 4349

ROYAL SIGNALS & RADAR ESTABLISHMENT

PARALLELISATION OF A DYNAMIC PROGRAMMING ALGORITHM SUITABLE FOR FEATURE DETECTION

Author: P G Ducksbury

PROCUREMENT EXECUTIVE,
MINISTRY OF DEFENCE,
R S R E MALVERN,
WORCS.

RSRE MEMORANDUM No. 4349

UNLIMITED

CONDITIONS OF RELEASE

0066653 BR-11330

********** DRIC U

********** DRIC Y

Reports quoted are not necessarily available to members of the public or to commercial organisations.

R.S.R.E. Memorandum 4349

Parallelisation of a Dynamic Programming algorithm suitable for feature detection

P. G. Ducksbury
Royal Signals and Radar Establishment.
St Andrews Rd, Malvern,
Worcs., WR14 3PS, UK.

January 1990

Abstract

This paper describes the approaches that were taken to produce a parallel algorithm that would be suitable for the problem of feature detection. The Full Image Search (FIS) algorithm which is based upon the Dynamic Programming technique was chosen as being the most suitable starting point for development on a multiprocessor system.

The concepts behind the Dynamic Programming algorithm are briefly introduced followed by a description of the different types of inherent parallelism that exist in the technique. A discussion then follows on which is the most suitable form of parallelism and how it can be effectively implemented on an array of transputers. Finally results are given which justify the time spent on this work together with ideas for future extensions to the work.

Contents

1 Introduction

Feature detection is a vital component of most image processing systems which must be performed with both accuracy and speed. In this paper the particular feature detector that is of interest is the location of wheels in urban and semi-urban settings.

The objective of this work is therefore to develop the ability for exploiting the vast potential that is available through the use of multiprocessor systems to enable both fast and reliable feature detection. Section 2 of this paper briefly describes the concept of the Dynamic Programming technique and then goes on to describe how this can be used in conjunction with the Full Image Search algorithm. The various forms in which the inherent parallelism of the Dynamic Programming algorithm exist are then discussed. The following section then describes some detail on the target hardware which is the transputer and how the transputers features of concurrency and processor to processor communication can be utilised to achieve the objective.

2 Dynamic Programming Topics

2.1 The Technique

The Dynamic Programming technique is based upon the principle of optimality and was introduced by Bellman [Bell 57] for the solution of certain classes of non-linear optimisation problems.

Let us now consider the following multi-stage optimisation problem

$$\min \sum_{i=1}^{N} g_i(x_i, u_i) \tag{1}$$

subject to the following constraint

$$x_{i+1} = f_i(x_i, u_i)$$

where

$$\left.\begin{array}{l} x_i \in X_i \\ u_i \in U_i \end{array}\right\} i = 1, 2, .., N-1$$

here X_i is defined to be the state set, U_i is defined to be the decision (or the control set), f_i is the transition (or production) function and g_i is the cost function.

Then the transition function f_i relates a state and a decision at a given stage i to the succeeding stage $i+1$. The cost function g_i gives the cost of taking a particular decision at a given state and stage.

The problem of minimising equation 1 over all stages subject to the constraints can be simplified using the principle of optimality (see Dixon [Dixon 72]) which states that any sub path of an optimal path is itself optimal. The calculation involved in the Dynamic Programming algorithm can now be reduced to the following recurrance relation which involves the calculation of an optimal value function V.

$$V_i(x_i) = \min_{u_i \in U_i} [g_i(x_i, u_i) + V_{i-1}(f_i(x_i, u_i))] \quad (2)$$

This effectively states that the optimal value for a sequence of i stages is expressed in terms of its value for preceeding *i-1* stages and its value at stage i.

The relation described in 2 is somtimes refered to as the "forward pass" of the Dynamic Programming algorithm, as opposed to the "backward pass" stage which generates the paths that have been obtained using the forward pass. We shall in this article only concern ourselves with possible ways of parallelising the forward pass.

2.2 Full Image Search Algorithm

The FIS algorithm has been reported elsewhere in detail by Series (see [Series 89] who provides a comparison of three different feature detection procedures). For the purposes of this description a path will be defined to be a set of adjacent pixels which provide a good correspondance with the reference model (*in this work the model considered is that of a wheel*).

Briefly the algorithm allows an optimal path to move over any pixel in the image, this being limited only by the choice at each stage of an entry from the set of productions. The match that is obtained between the image and the reference model is composed of two sets of terms. The first are the production penalties which relate the distortions suffered by a path in the image. The second are the local costs which describe the distance between a point in the image and that in the reference, these local costs being calculated from the gradient intensity of the image and the reference model.

The advantages of using the FIS algorithm are that as it is based on the Dynamic Programming technique it is able to deal with different complex reference shapes without altering the structure of the algorithm, in addition to this it is possible to train the algorithm using for example the Viterbi method and hence improve the efficiency of the feature detector.

2.3 Parallelism in Dynamic Programming

Despite the advantage that Dynamic Programming provides it will be seen that there is still a considerable amount of computation involved in the calculation, this will obviously severeley limit its use in certain time critical applications.

The introduction of relatively cheap microprocessors has meant that special parallel architectures can be constructed for what would be considered a very nominal cost. These machines are considered to provide the key to the improvement in performance that is required from the Dynamic Programming algorithm.

There have been a number of papers published which have concentrated on the different types of parallelism that are inherent inside the Dynamic Programming algorithm, for examples of these see [Dabass 80], [Casti 73], [Bert 84]. For a general

paper describing parallel architectures that are suitable for the different levels of machine vision see [Sanz 89].

If we now consider the Dynamic Programming algorithm then we can see that it consists essentially of three loops, as follows

```
Iterate for each stage (N)
    Iterate for each state (S)
        Iterate for each decision (U)
            Evaluate equation 2
```

As [Dabass 80] correctly points out, that as equation 2 is of a recursive nature then there would be no advantage in allocating each stage of the optimisation to a seperate processor as V_i clearly requires the computation of V_{i-1}.

There are therefore two main methods for partitioning the algorithm which we will consider, namely parallel control-space and parallel state-space.

2.4 Control-Space Parallelism

In the control-space method each processor has allocated to it a small region of the control (or decision) space. Therefore each processor will generate a local optimum to equation 2 for its own limited control space, but over all of the state space.

The disadvantage of this becomes apparent, as at the end of each stage some master process must then recieve all of these local optima and compare them to produce a global optima before the next stage can start. As the state space is likely to be very large we can anticipate that there will be a requirement for a large amount of communications in this approach.

2.5 State-Space Parallelism

In the state-space method however each processor is allocated a small region of the state space. As each processor now handles the entire control space the result will be a global optimum for its particular region of state space. Hence there is no requirement for any further comparisons to be made by a master process which would obviously introduce delays into the system. The second advantage of this approach is that each processor now only requires a reduced amount of information to be communicated at each stage.

We would naturally expect the latter method to be the most suitable, certainly for a MIMD type machine which uses local memory and communicates via message passing.

For readers who are interested in more information on the above two approaches [Dabass 80] describes them in greater detail and also contains discussions on the communications issues. For more detailed information on the communications issues see [Lint and Ager 81].

3 Implementation of State-Space Parallelism

3.1 Target Hardware

The target hardware that is being used for this work is the transputer with the implementation language being Occam. Figure 1 illustrates the simple hardware arrangement, each processor in the array is a T800 with 1 MByte of memory, whilst the transputer resident in the host PC is a T414 with 2MByte of memory. As can be seen transputer 0 is being used as an interface between the array and the host PC with the additional task of being responsible for input of the initial image and output of the final results.

The array is arranged in a simple pipeline structure with input to transputer 1 and output from transputer 16. However by using the fact that the transputers links are bi-directional we effectively have a two way ring with input to either transputer 1 or transputer 16 and similarly for the output. This two way structure being exploited to the full as will be described latter.

3.2 Approach for Parallelisation

There are a number of different methods for decomposing algorithms into a form suitable for parallelisation, such as

- Task (or processor) Farm : Each processor executes an identical copy of the program *in isolation* from the remaining processors.
- Geometric Parallelism : This can be considered as an extension to the processor farm approach. Each processor executes an identical copy of the program on data which is a subregion of the problem and then communicates boundary data to neighbouring processors.
- Algorithmic Parallelism : Each processor is responsible for a part of the algorithm and all data passes through each processor's code.

In this case it is the geometric parallelism that is considered to be the most suitable approach. *In actual fact this approach could be applied to either the state-space or the control-space methods, as both rely on partioning the data and having the same code on each processor.*

Figure 2 shows the top level of a particular slave process which will be resident on every processor in the array, in it we see that there are two processes running in parallel. One of these is responsible for receiving data from either of the previous processors in the network and then either routing it to the calculator process for processing or onto either of the next processors in the network if the calculator is busy.

Given our description of state-space parallelism and considering the structure of the array the approach adopted will therefore be for each slave processor to be

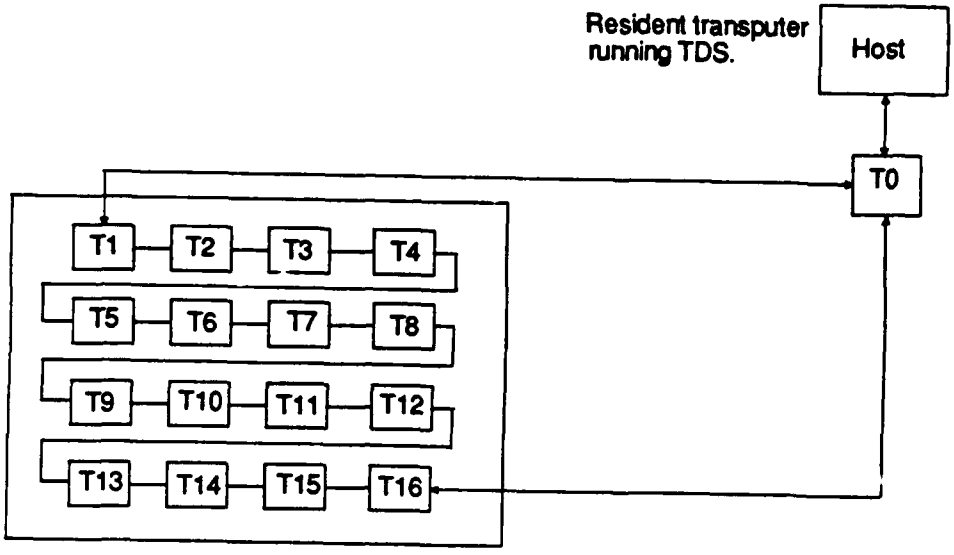

Figure 1: *Typical Hardware layout*

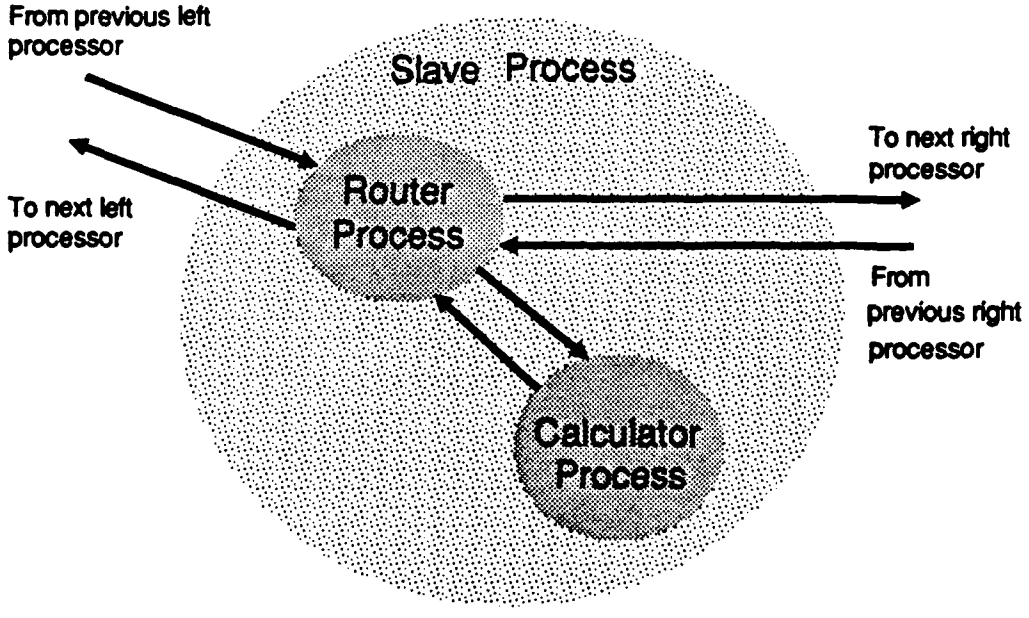

Figure 2: *Slave process*

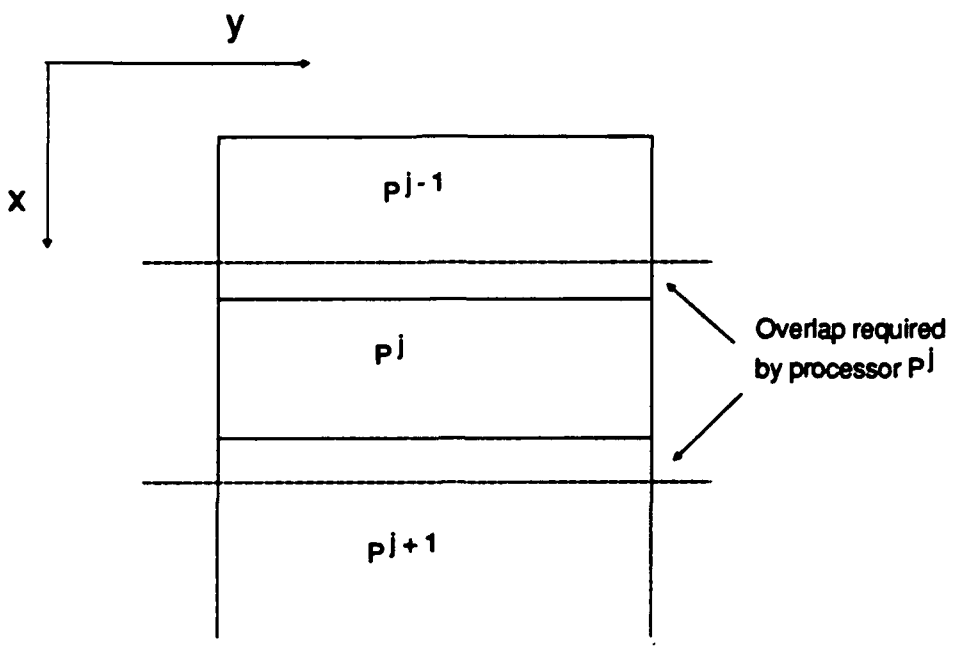

Figure 3: *Processor Boundary layout*

given responsibility for a $\frac{1}{16}$ rectangular portion of the state space. (*An alternative mapping is described in Appendix A*) There is now one problem that needs to be overcome, that is the boundary communication problem. Given that processor j is calculating V_i^j at stage i it requires not only its previous information, namely V_{i-1}^j but also that of its neighbours ie. V_{i-1}^{j-1} and V_{i-1}^{j+1}, as can be seen from figure 3.

This common information can be provided to the processors in one of two possible ways as follows.

3.2.1 Master/Slave Communication

In this case the communication is achieved by each transputer sending its appropriate overlaps back to the master transputer at the end of every iteration. (ie *using a one way network*) The master transputer then has the sole responsibility for ensuring that each slave transputer receives the correct information at the start of the next iteration. Using this method however we can see that to communicate all data to the master and back to the slaves requires the use of the order of $2\sum_{i=1}^{P} i$ links. (*where for $P = 16$ this means the use of approximately 272 links*)

3.2.2 Slave/Slave Communication

In this case the communication is achieved by utilising the two way structure of the array. Each transputer shifts its data to its right neighbouring processor and then shifts the data to its left neighbouring processor. Thus the same effect has been achieved in the use of $2(P-1)$ links. This approach has another advantage in that

Figure 4: *Top left to bottom right. Initial image, accumulated merit, angular data of initial image and the accumulated merit at the mid-point of all paths*

once the array has been *booted* by the master transputer there is no longer any need for communication between the two, as we could anticipate that this would soon become a bottleneck.

The former method of Master/Slave communication would indeed be acceptable on a network with a small number of transputers but clearly becomes unacceptable on a larger network. Hence the latter approach is to be the one that is adopted for this work.

4 Results

Figure 4 shows the typical output from the algorithm where for both the accumulated merit and mid-point display the white areas denote regions of high merit (*or activity*). As the paths generated will focus in on the strong features of the image the end points will show a characteristic diffuse pattern. The mid-points of these paths will however be strongly clustered which can be seen from the very sparse structure of the midpoint display.

The typical execution times obtained from running the Dynamic Programming

Case	*Sequential Processor*	*Host transputer*	*Multi transputer*
i	313	202	29
ii	n/a	n/a	16.5

Table 1: *Processor Performance figures (Seconds)*

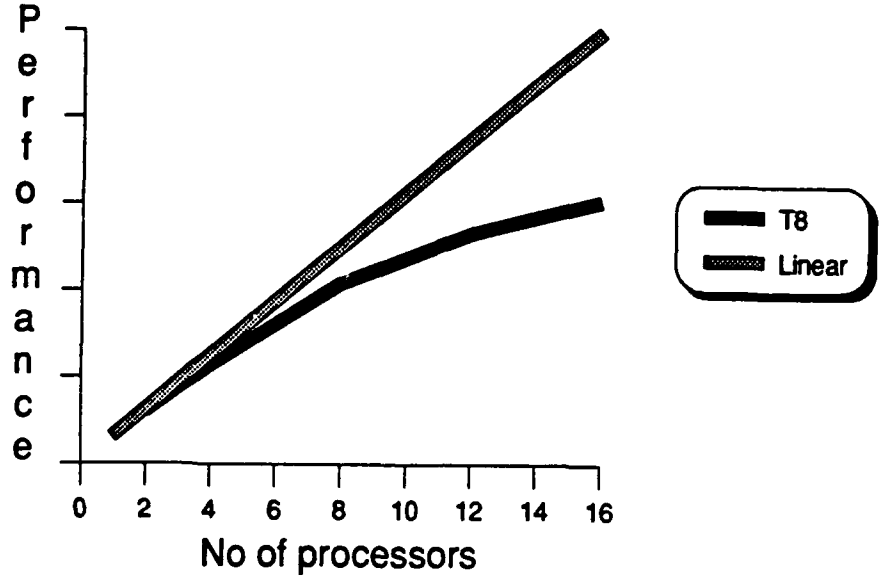

Figure 5: *Performance (1/cpu time) v Number of processors*

(forward pass) algorithm are given briefly in Table 1, while figure 5 shows the speed-up obtained with an increasing number of processors

The following points should be noted regarding the results

- All timings are in seconds with all runs using a 128 x 128 image with a 32 stage model.
- The Sequential version used for comparison is in FORTRAN running on a DEC VAX 11/780.
- Case (i) is using the one way Master/Slave network approach.
- Case (ii) is using the two way Slave/Slave network approach.

As we would expect the two way approach is superior as once the initialisation has been performed there is no longer a communications bottleneck with the host transputer. The relationship from figure 5 clearly starts to significantly loose its linearality as more than 8 processors are included in the system.

However we should not loose sight of the fact that with the full network of 16 processors the achieved speed-up of approximately 19 times the VAX [1] is a significant and acceptable starting point for this work, further expansion of this point can be found in the next section.

5 Summary and Future work

This article has briefly described the Dynamic Programming algorithm and how it can be sucessfully designed to run on a multi-transputer array. The results obtained are comparable with that of the sequential version but have the added advantage of the speed improvements that can be gained from the use of a powerfull multi-transputer array.

There are a variety of ways in which this work can be extended some or all of which will be considered in the future

- Extensions to incorporate a four-way communications structure for the broadcasting of the boundary data. (*This may reduce some of the idle time in the processors, See Apppendix A for details*)
- Extension to run on larger transputer arrays. This should now be achievable with only reconfiguration and parametric changes to the Occam code.
- Extension to larger images than 128 x 128. This should be possible as with multiple transputers the large memory requirement can be distributed.
- Parallelisation of the Backward Pass of the algorithm. This can currently trace back the path for every pixel in the image in approximately 10 seconds on the host transputer.

References

[Bell 57] Bellman R. E. *"Dynamic Programming"*, Princeton University Press, Princeton, 1957.

[Dixon 72] Dixon L. C. W. *"Nonlinear Optimisation"*, English Universites Press, 1972.

[Series 89] Series R. W. et.al. *"Comparison of Approaches to Feature Detection"*, Alvey Vission Conference, University of Reading 9/89.

[Dabass 80] AL-Dabass D. *"Two methods for the solution of the Dynamic Programming algorithm on a multiprocessor cluster"*, Optimal Control Applications and Methods, vol. 1, pp 227-238 1980.

[1] This can only be an approximate estimate due to the difficulty in assessing the speed of a multiuser system

[Casti 73] Casti J. et.al. *"Dynamic Programming and Parallel Computers"*, Journal of Optimization Theory and Applications, vol. 12, no. 4, 1973.

[Bert 84] Bertolazzi P., Pirozzi M. *"A Parallel Algorithm for the Optimal Detection of a Noisy Curve"*, Computer Vision, Graphics, and Image Processing, vol. 27, pp 380-386, 1984.

[Lint and Ager 81] Lint B., Agerwala T. *"Communication Issues in the Design and Analysis of Parallel Algorithms"*, IEEE Trans. on Software Engineering, vol. SE-7, no. 2, pp 174 - 188, March 1981.

[Sanz 89] Sanz J.L.C. *"Which Parallel Architectures are useful/useless for Vision Algorithms?"*, Machine Vision and Applications, vol. 2, pp 167 - 173, 1989.

A Alternative Data Mappings

The method that has been described in this paper of giving each processor a number of rows of the data space may not appear to be the most efficient. An alternative exists and is described below in comparison with the one used.

The data space could have been subdivided up into small squares rather than long rectangles. Although the size of the data space would remain the same the perimeter (*or boundary*) area would be less. This approach would mean that data would not only need to be shifted east and west (*as is the case with the two way communications approch*) but also north and south, thus we would be utilising a four way communications approach which would mean that the data would not have to travel over as many processors to reach its ultimate destination. To consider this further, let us calculate approximately the amount of data communicated and time taken for both cases. First we need to state a few assumptions

- The image we are dealing with is 128 x 128 in size.
- There are 16 processors in the system.
- We can ignore the boundary around the complete data space as this is initialised and is never updated during the algorithm.
- Each packet of the data space requires a boundary of width 2 and 32 bit arithmetic is being used.

For the two-way communications we have
2(P-1) x (2x128) x 4 bytes/real number = 30.7 KBytes
and for the four-way communications we have
3P x (2x(32+2+2)) x 4 bytes/real number = 13.8 KBytes
Total saving = 16.9 KBytes

In the case of the four-way communication the term 3P is derived from the following $\sum n(n-1)$ where the summation is performed over the north, south, east and west directions and n is the number of squares in the x and y directions.

If we now assume that are links operate at 20MBytes/s then we are only talking of a saving of 0.85ms/iteration. If the realistic figure for the links was nearer to 10MBytes/s then this saving would be 1.7ms/iteration. This appears to be only a very negligable saving that would obviously have to be considered against the extra time that would be required to produce the code for the four-way communications structure. [2]

[2]However it should be noted that this time is the theoretical communication time saving over the actual links. It does not attempt to measure the idle time in the individual processors (*due either to waiting for data or general housekeeping*) thus it may be desirable to consider this approach in the future.

DOCUMENT CONTROL SHEET

Overall security classification of sheet ..... Unclassifed .....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification, eg (R), (C) or (S))

1. DRIC Reference (if known)	2. Originator's Reference	3. Agency Reference	4. Report Security Classification
	Memo 4349		Unclassified

5. Originator's Code (if known)	6. Originator (Corporate Author) Name and Location
7784000	ROYAL SIGNALS & RADAR ESTABLISHMENT ST ANDREWS ROAD, GREAT MALVERN WORCS WR14 3PS

5a. Sponsoring Agency's Code (if known)	6a. Sponsoring Agency (Contract Authority) Name and Location

7. Title

PARALLELISATION OF A DYNAMIC PROGRAMMING ALGORITHM SUITABLE FOR FEATURE DETECTION

7a. Title in Foreign Language (in the case of Translations)

7b. Presented at (for Conference Papers): Title, Place and Date of Conference

8. Author 1: Surname, Initials	9a. Author 2	9b. Authors 3, 4 . . .	10. Date	pp. ref.
DUCKSBURY P G			1990.01	13

11. Contract Number	12. Period	13. Project	14. Other Reference

15. Distribution Statement

Unlimited

Descriptors (or Keywords)

Continue on separate piece of paper

Abstract

This paper describes the approaches that were taken to produce a parallel algorithm that would be suitable for the problem of feature detection. The Full Image Search (FIS) algorithm which is based upon the Dynamic Programming technique was chosen as being the most suitable starting point for development on a multiprocessor system.

The concepts behind the Dynamic Programming algorithm are briefly introduced followed by a description of the different types of inherent parallelism that exist in the technique. A discussion then follows on which is the most suitable form of parallelism and how it can be effectively implemented on an array of transputers. Finally results are given which justify the time spent on this work together with ideas for future extensions to the work.